Implementation of the HotNet2 Network Diffusion-based Analysis Method in Java

By

Melissa Yan, B.S.

Master's Capstone Project

Presented to the
Division of Bioinformatics & Computational Biology
Department of Medical Informatics & Clinical Epidemiology
and the
Oregon Health & Science University
School of Medicine
in partial fulfillment of the requirements of

Master of Biomedical Informatics

December 2016

School of Medicine

Oregon Health & Science University

CERTIFICATE OF APPROVAL

This is to certify that the Master's Capstone Project of

MELISSA Y. YAN

"Implementation of the HotNet2 Network Diffusion-based Analysis Method in Java"

Has been approved

Guanming Wu, Ph.D.
Capstone Advisor
Department of Medical Informatics and Clinical Epidemiology

Table of Contents

Acknowledgements

I would like to thank God for the countless blessings He has given me and for providing me this opportunity to pursue and finish my master's degree. It has been a great privilege to work under the guidance of my advisor Dr. Guanming Wu. I deeply appreciate his patient continuous support and encouragement. I would also like to thank Dr. Shannon McWeeney and Dr. Beth Wilmot for providing me advice throughout my project. I am also very grateful for the help I received from Ms. Diane Doctor. Finally, I would like to extend my appreciation towards my friends and family for their love and for always believing in me.

Abstract

Network-based approaches are widely used for analyzing large-scale genomic data and searching for disease driver genes. Diffusion-based approaches are one of the most popular approaches for detecting disease candidate genes by combining biological information with topological network information. A particular diffusion-based algorithm used in The Cancer Genome Atlas data analysis called HotNet2 has been able to detect significant subnetworks of genes related to various cancer types. The purpose of this capstone project is to port HotNet2 from Python to Java. This will allow future integration into Java-based standalone projects that can provide users a more interactive user interface for the whole HotNet2 workflow.

1. Introduction

High-throughput experiments are known for producing massive complex biological data sets. Although most of these data sets are readily available to the public, they often only contain partial omics information and noisy data with false positives or negatives.[1] Consequently, researchers are faced with the challenge of how to process, extract, and explain relevant biological information to gain a better understanding of diseases and many biological processes using these massive data sets.[2,3]

One popular and powerful computational approach for performing data analysis is to combine omics data with networks to extract and explain patterns related to biology. Networks have provided new insights and are moving the bioinformatics field forward. They have been used to aid in predicting protein functions,[4] comprehending chemical reactions in organisms,[5] understanding regulation of gene expression,[3,6–8] and detecting relationships between diseases.[9,10] For instance, Figure 1 illustrates that using a network allows detection of relationships between different diseases through overlapping genes in a protein-protein interaction network. To easily inspect relationships, it is possible to inspect a group of diseases by adding or removing certain diseases overlaid onto the network.

Network-based approaches used to analyze omics data often use protein-protein interaction networks.[2,11,12] Those interaction networks are comprised of nodes, which represent genes or proteins, and edges, which represent existing interactions between genes or proteins. Networks often are stored in the form of various matrices in software applications (Figure 2). Using matrices allows computational tools to perform matrix arithmetic for data analysis, pattern detection and integration of additional biological data.

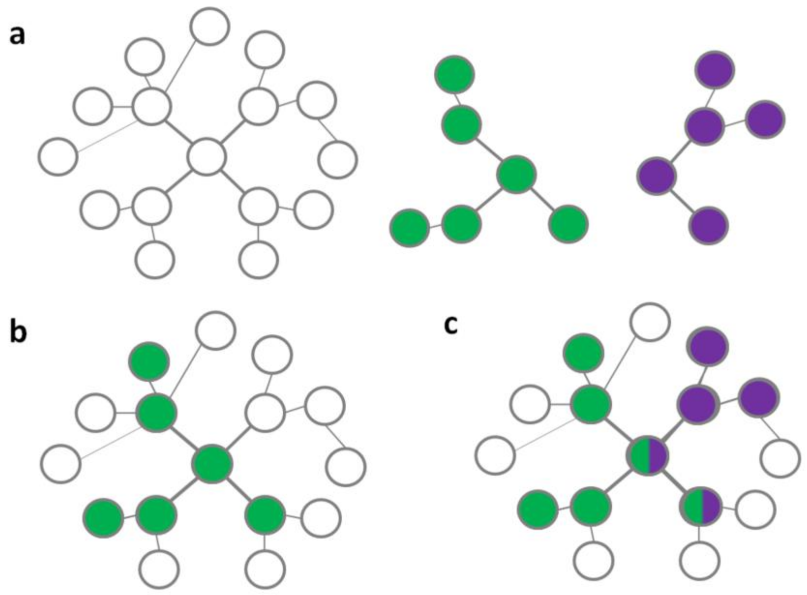
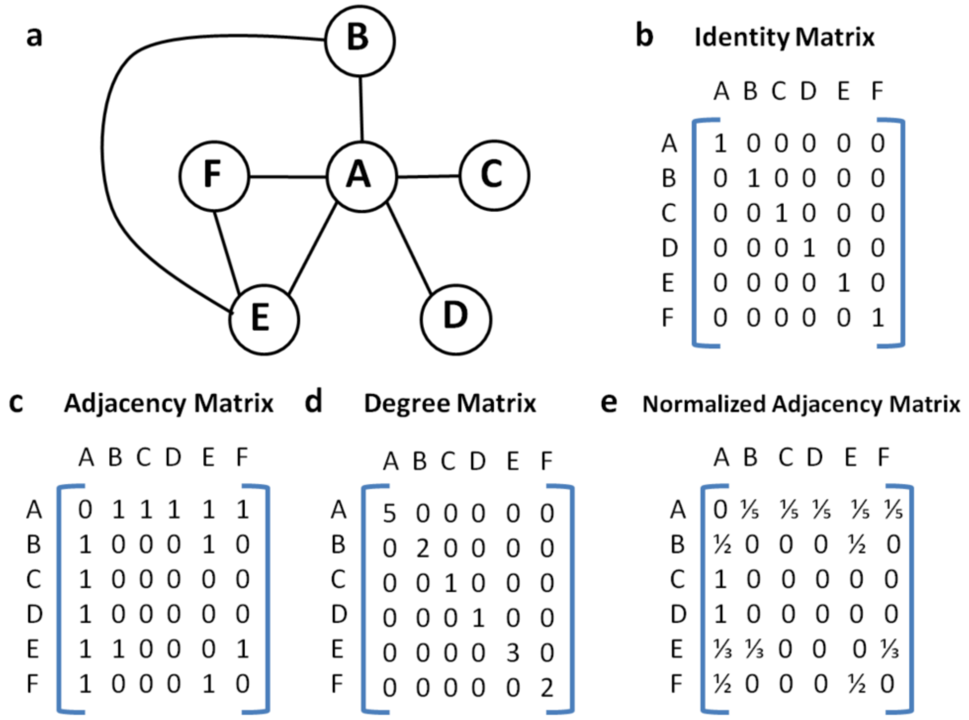


Figure 1. *Mapping diseases onto a protein-protein interaction network. (**a**) The human interactome network is the gray network and two disease gene groups or modules are represented in green and purple. (**b**) Only the green disease module is integrated into the network. (**c**) After mapping both disease associated gene modules onto a network, there are 2 nodes with green and purple colors which indicate overlapping genes in both diseases and therefore the similarity of these two diseases. *Note: The figure is a modification of Figure 3 iii from Barabási, Gulbahce, and Loscalzo 2011.*[10]



$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 1 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 1 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 1 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 1 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 1 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 1 & 1 & 1 & 1 & 1 \\ B & 1 & 0 & 0 & 0 & 1 & 0 \\ C & 1 & 0 & 0 & 0 & 0 & 0 \\ D & 1 & 0 & 0 & 0 & 0 & 0 \\ E & 1 & 1 & 0 & 0 & 0 & 1 \\ F & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 5 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 2 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 1 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 1 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 3 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 2 \end{array}$$

$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ B & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ C & 1 & 0 & 0 & 0 & 0 & 0 \\ D & 1 & 0 & 0 & 0 & 0 & 0 \\ E & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ F & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \end{array}$$

Figure 2. *A network and various matrices related to it.* ***(a)*** *A network with 6 nodes of A, B, C, D, E, F and 7 edges.* ***(b)*** *An identity matrix is used to keep track of nodes within a network. It forms a diagonal of 1s and assigns 0 to the rest of the matrix. For instance, since node A exists in the network, a value of 1 is assigned to the matrix at row A and column A.* ***(c)*** *An adjacency matrix stores information about edges within a network. If an edge exists, then a value of 1 is assigned, otherwise the value is 0. For node E, there are 3 edges formed by E-A, E-B, and E-F, so in row E the columns A, B, and F are assigned the value of 1 and the remaining columns are assigned 0.* ***(d)*** *A degree matrix contains diagonal information about the number of edges a node has. Node A has 5 edges, so the value of 5 is recorded in the matrix at row A and column A.* ***(e)*** *A normalized adjacency matrix is calculated by dividing the individual values of the adjacency matrix by the degree matrix's diagonal row values. Each value in the adjacency matrix for row E is divided by the degree matrix's value of 3 in row E column E, so the results in the normalized adjacency matrix are ⅓ and 0.*

Interpreting biological data in the past frequently focused on individual components instead of integrating multiple components within and between complex systems of genes, proteins and other gene products, cells, organs, and organisms.[13–16] For instance, prior disease studies only focused on an individual gene related to a single disease. Now, studies are more focused on the holistic approach of inspecting multiple mutated genes within a disease and the relationship those mutated genes have. To study multiple mutated genes and their relationships to diseases, network-based approaches often find modular structures, called modules, in biological networks.[1] These modules are often used to find disease related genes or proteins, discover biomarkers, detect drug-targeted proteins and pathways, identify evolutionary conservation, and predict protein functions.[1,17]

One of the goals in disease related studies is to use disease modules to identify novel disease gene candidates. The diffusion-based method is one of best methods in predicting disease candidate genes because besides integrating biological data, it also incorporates information related to topological and functional modules within the whole network.[10,18–20] This method is based on the concept of diffusion in a network setting using gene scores. Gene scores measure the significance of how likely a gene will cause the disease to occur if mutated. For instance, gene scores can be mutation scores obtained from the frequency of gene mutations within tumors (refer to “sample use case” below for an example based on frequency of gene mutations). Diffusion begins at each gene and each gene score slowly spreads out towards other genes in the protein-protein interaction network.[19–21] When diffusion reaches to equilibrium, a network module detecting algorithm is used to find network modules having major distributions of the gene score. Genes in these modules are assumed to be disease related, measured by statistical tests usually based on random permutation.[10,21,22]

1.1 Sample Use Case

Diffusion-based algorithms are widely used in cancer studies. Cancer is a genetic disease. Mutations in cancer driver genes cause tumor development. A technique to detect cancer driver genes is to find significantly mutated genes within patient tumors.[23] Although some cancer driver genes have a high frequency of mutation, most of them have much lower frequencies that are not deemed statistically significant, causing a long-tail distribution of infrequent mutations in cancer driver genes, known as the long-tail phenomenon.[23–26] In Figure 3, highly significantly mutated genes such as TP53 and KRAS are shown in red on the left side of the distribution curve and insignificantly mutated genes are blue in the distribution tail. Although BRCA1 is a known cancer driver gene in breast cancer, large-scale mutation screening performed by Van Allen et al showed that BRCA1 did not carry significantly frequent mutations and ended up on the tail of the distribution curve.[23] By placing infrequently mutated genes together with frequently mutated genes in the network context, network-based approaches can increase the analysis power to detect cancer driver genes with lower frequent mutations. HotNet2 is one of the network-based approaches used to detect cancer driver genes with low mutation frequencies by searching for significant cancer-related network modules.

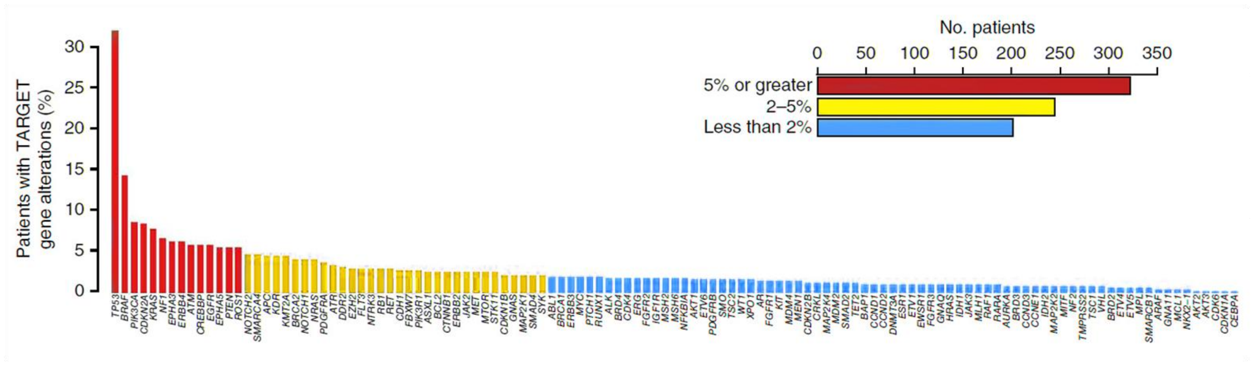


***Figure 3**. Frequency of mutated genes from patients in a study done by Van Allen et al., 2014. The figure is a modification of Figure 3f by Van Allen et al..* [23]

<u>1.2 HotNet</u>

To better understand HotNet2, a background on the previous version known as HotNet is provided here. HotNet is based on the concept of heat diffusion, the process by which heat disperses from a source and warms up the surroundings.[27,28] In HotNet, mutated genes are considered "hot" and their heat will spread through the network and impact other genes in the network. Genes close to the hot genes have a higher chance of obtaining more heat from hot genes than genes further away (Figure 4a and 4b). Also, genes near hot genes with few neighbors have a higher chance of obtaining more heat than genes near hot genes with many neighbors, as seen in Figure 4c and 4d.

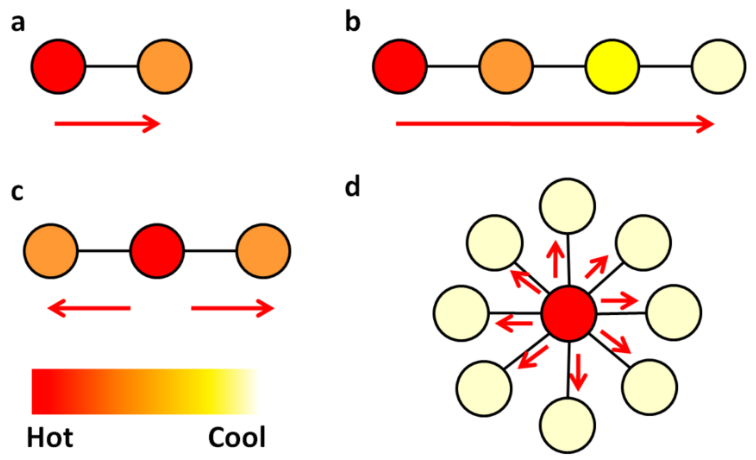


Figure 4. *The diffusion process modeled with different scenarios with the heat scale of red as hot and white as cool.* (***a***) *Heat from the hot left red node is diffusing to the neighboring node on the right.* (***b***) *As the heat from the left node diffuses towards the right, the amount of heat transferred lessens. Nodes further away from the source of heat receive less heat than those closer.* (***c***) *Heat from the middle red node is shared between the 2 orange nodes on the side.* (***d***) *The middle heat is distributed to the nearby 8 surrounding nodes. The more neighbors a heat source node has the less heat each neighbor has because the heat is evenly shared with the surrounding nodes.*

There are 4 steps in HotNet.[27,28] The first is to determine how all pairs of genes in a network influence or affect one another based on diffusing heat in the network. Using heat diffusion, the amount of heat a gene in the network receives from a mutated gene after a certain time is obtained for the influence estimation. In Figure 5a, after the first time of diffusion node B receives all the heat from mutated node A. However, in Figure 5b, after the first time of diffusion node A receives only 1/5 of the heat from mutated node B. This process is repeated until an equilibrium state is reached and recorded as the influence estimation of all genes within the network. The second step places actual gene scores onto the network based on influence estimation obtained in the previous step. Then the least amount of heat transferred between a pair of nodes is recorded as the heat transfer of both nodes (Figure 5a-c). The same process is repeated for all pairs of nodes to estimate how genes will impact each other in a network. Genes with greater significance will have higher scores that result in higher temperatures. In the third step, all gene to gene relationships below a certain heat threshold are removed to enable formation and discovery of subnetworks. In the final step, statistical tests are used to determine if subnetworks are significant. For all possible subnetwork sizes p-values are obtained based on the numbers of subnetworks for a given size for the actual network and randomly formed networks. From the p-values, HotNet calculates the false discovery rates (FDRs) of subnetworks and finds significant subnetworks.

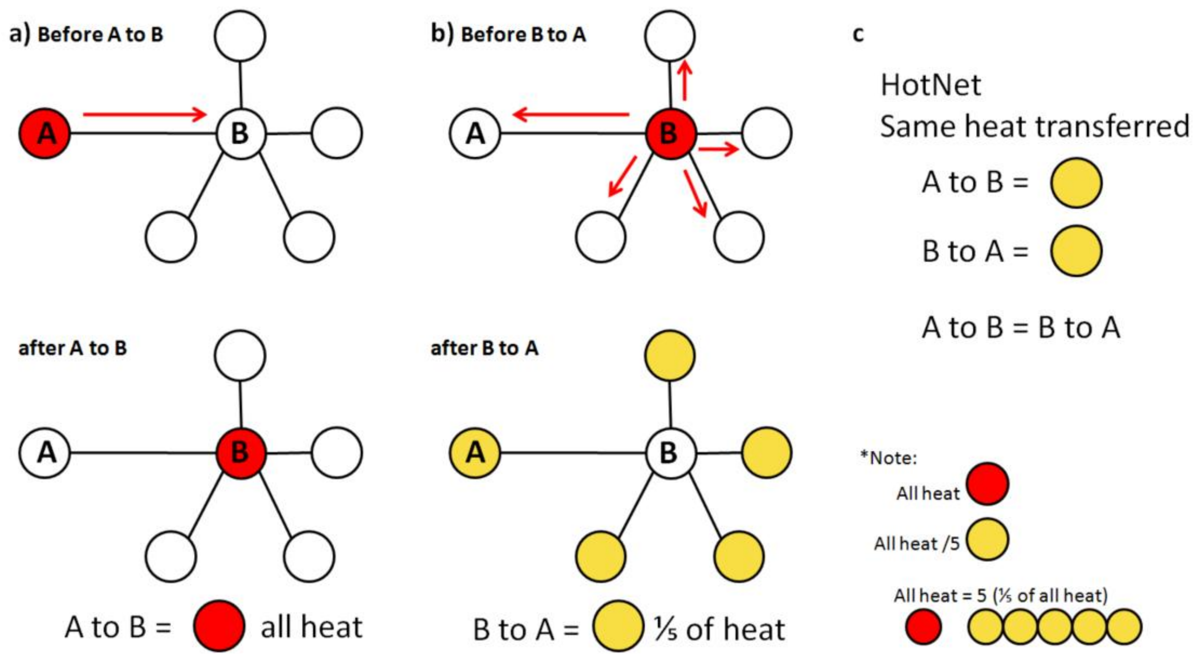
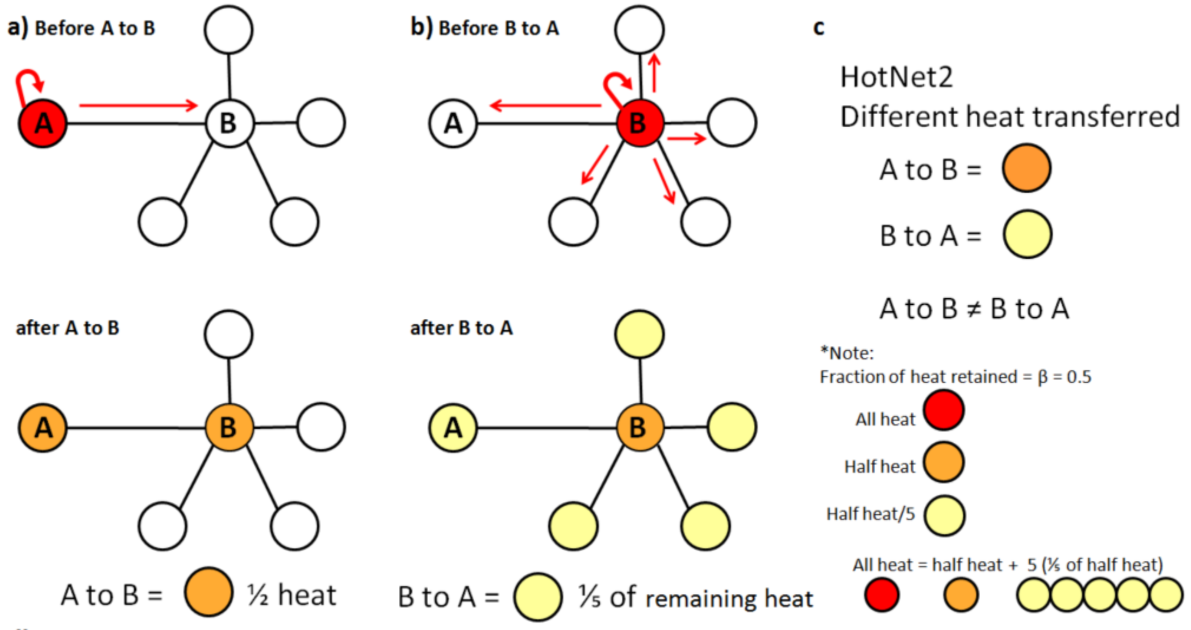


***Figure 5**. Illustration of HotNet. In HotNet, to determine influence between gene pairs, all the heat is transferred to neighbor(s) using heat diffusion. (**a**) Node A only has 1 neighbor, so all the heat from A is diffused into B. (**b**) Node B has 5 neighbors, which all receive equal portions of the heat from B. Node A receives ⅕ of the heat from Node B. (**c**) In HotNet, the heat transferred between nodes A to B and B to A is ⅕ of the total heat. The amount of influence between gene pairs is solely based on the smallest amount of heat transferred between the pair. It disregards the direction heat travels from.*

Although HotNet was able to find subnetworks in actual TCGA Pan-Cancer mutation data, there is a high likelihood that many of those subnetworks are false positives, because many of these subnetworks were biased towards star shaped clusters that also frequently showed up in the random permutations.[29] These star shaped clusters have a hot central node surrounded by many cold nodes (Figure 4d). The bias occurs because HotNet does not consider the direction of heat from the mutated genes. To address this issue, the HotNet2 algorithm was developed.

1.3 HotNet2

Instead of using heat diffusion without considering heat directionality, HotNet2 uses a different type of diffusion and incorporates the impact of heat directionality. The first step of HotNet2 uses insulated heat diffusion, allowing nodes to retain some of their own heat in the process of transferring heat to neighboring nodes (Figure 6a, b).[29] In the second step, to determine how genes influence each other, instead of recording the least amount of heat transferred for both genes as done in HotNet (Figure 5c), HotNet2 incorporates information about the direction of heat traveling between nodes for the two genes separately (Figure 6c). After these two initial steps, the remaining steps in HotNet2 are similar to HotNet. Examples of the first three steps of HotNet2 are shown in Figure 7, 8, and 9.



d)

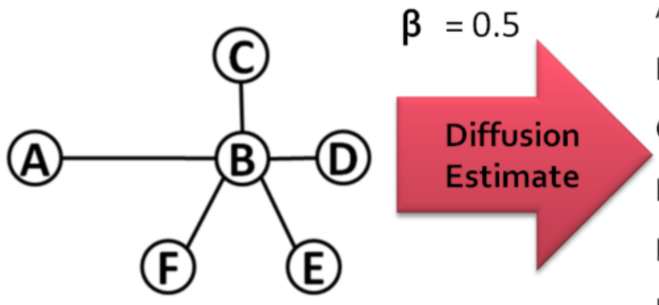
$$F = \beta\,(I - (1-\beta)W)^{-1}$$

F = heat diffusion matrix
β = heat retained by each node
I = identity matrix
W = normalized adjacency matrix

***Figure 6**. Illustration of HotNet2. In HotNet2, to determine influence between gene pairs, the heat is transferred to neighbor(s) and some heat is retained by each node using insulated heat diffusion. The amount of heat retained by each node is dependent on insulating parameter β, which provides the fraction of heat each node retains. Here β is 0.5, so half the heat from each node is retained and the remaining half is shared evenly among the node's neighbor(s). (**a**) Node A only has 1 neighbor, so half the heat is kept in A and the remaining half is diffused into B. (**b**) Node B has 5 neighbors, so half the heat is kept in B and the remaining half is distributed in equal portions among B's neighbors. (**c**) HotNet2 does not disregard the direction of heat transfer and treats the cases (**a**) and (**b**) separately because the direction of heat can result in discovery of subnetworks less biased towards star shaped and cooler subnetworks. (**d**) The insulated heat diffusion process can be described with a matrix-based equation, which contains parameter, β, for amount of heat retained by each node, an identity matrix, and a normalized adjacency matrix. Descriptions of matrices can be found in Figure 2.*

$$F = \beta\,(I - (1-\beta)W)^{-1}$$

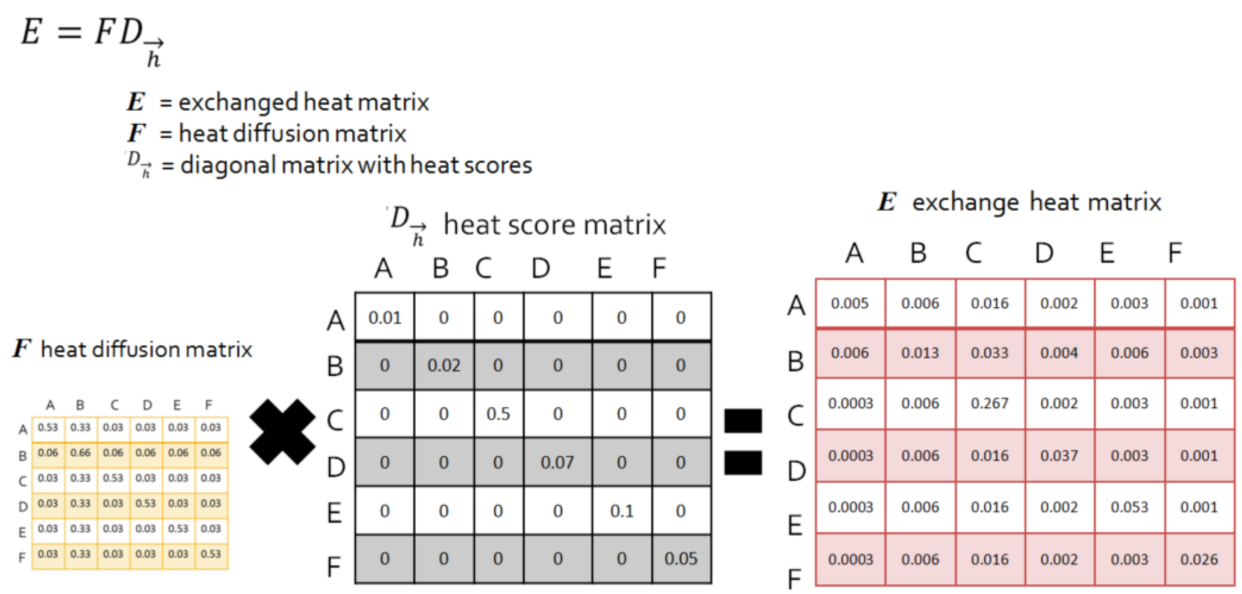
F	= heat diffusion matrix
β	= heat retained by each node
I	= identity matrix
W	= normalized adjacency matrix



F heat diffusion matrix

	A	B	C	D	E	F
A	0.53	0.33	0.03	0.03	0.03	0.03
B	0.06	0.66	0.06	0.06	0.06	0.06
C	0.03	0.33	0.53	0.03	0.03	0.03
D	0.03	0.33	0.03	0.53	0.03	0.03
E	0.03	0.33	0.03	0.03	0.53	0.03
F	0.03	0.33	0.03	0.03	0.03	0.53

***Figure 7.** The first step of HotNet2 described with a matrix based equation and example. From an interaction network with 6 nodes, by modeling it as a normalized adjacency matrix and performing the steps in the equation using the provided heat retention beta parameter, an insulated heat diffusion matrix is generated.*



$$E = FD_{\vec{h}}$$

E = exchanged heat matrix
F = heat diffusion matrix
$D_{\vec{h}}$ = diagonal matrix with heat scores

F heat diffusion matrix

	A	B	C	D	E	F
A	0.53	0.33	0.03	0.03	0.03	0.03
B	0.06	0.66	0.06	0.06	0.06	0.06
C	0.03	0.33	0.53	0.03	0.03	0.03
D	0.03	0.33	0.03	0.53	0.03	0.03
E	0.03	0.33	0.03	0.03	0.53	0.03
F	0.03	0.33	0.03	0.03	0.03	0.53

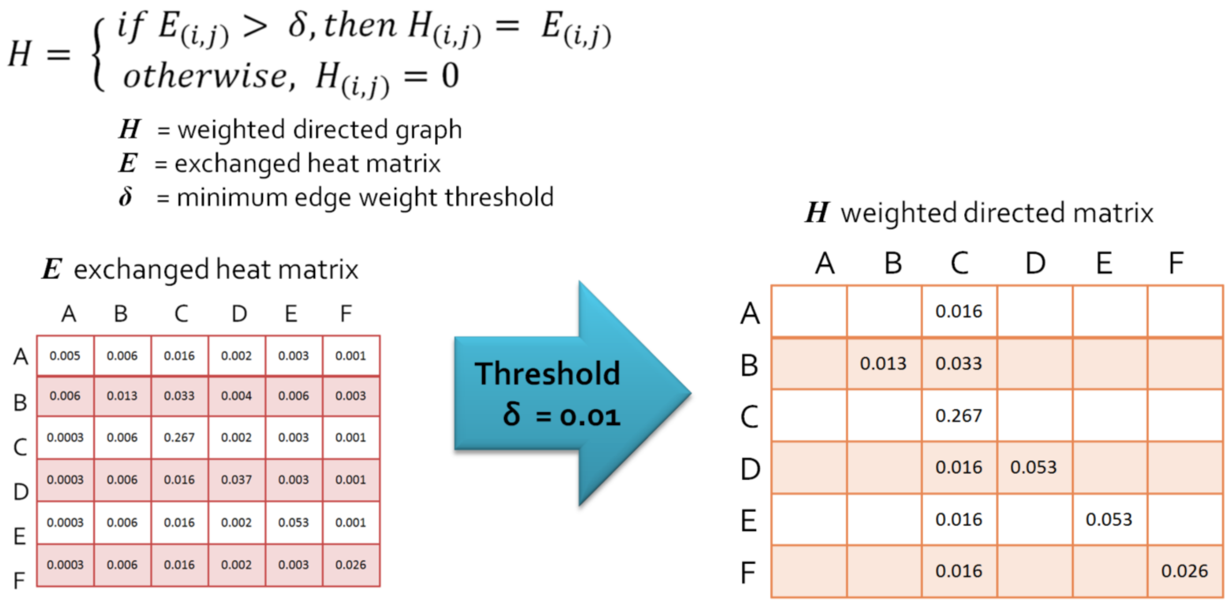
$D_{\vec{h}}$ heat score matrix

	A	B	C	D	E	F
A	0.01	0	0	0	0	0
B	0	0.02	0	0	0	0
C	0	0	0.5	0	0	0
D	0	0	0	0.07	0	0
E	0	0	0	0	0.1	0
F	0	0	0	0	0	0.05

E exchange heat matrix

	A	B	C	D	E	F
A	0.005	0.006	0.016	0.002	0.003	0.001
B	0.006	0.013	0.033	0.004	0.006	0.003
C	0.0003	0.006	0.267	0.002	0.003	0.001
D	0.0003	0.006	0.016	0.037	0.003	0.001
E	0.0003	0.006	0.016	0.002	0.053	0.001
F	0.0003	0.006	0.016	0.002	0.003	0.026

Figure 8. *The second step of HotNet2 enables the network to incorporate gene score information. The gray heat score matrix ($D_{\vec{h}}$) places gene scores onto the network by multiplying with the diffusion matrix from step 1 (Figure 7).*



$$H = \begin{cases} if\ E_{(i,j)} > \delta, then\ H_{(i,j)} = E_{(i,j)} \\ otherwise,\ H_{(i,j)} = 0 \end{cases}$$

H = weighted directed graph
E = exchanged heat matrix
δ = minimum edge weight threshold

E exchanged heat matrix

	A	B	C	D	E	F
A	0.005	0.006	0.016	0.002	0.003	0.001
B	0.006	0.013	0.033	0.004	0.006	0.003
C	0.0003	0.006	0.267	0.002	0.003	0.001
D	0.0003	0.006	0.016	0.037	0.003	0.001
E	0.0003	0.006	0.016	0.002	0.053	0.001
F	0.0003	0.006	0.016	0.002	0.003	0.026

H weighted directed matrix

	A	B	C	D	E	F
A			0.016			
B		0.013	0.033			
C			0.267			
D			0.016	0.053		
E			0.016		0.053	
F			0.016			0.026

Figure 9. *The third step of HotNet2 enables the formation of hot subnetworks. Using the exchanged heat matrix from the second step (Figure 8), the minimum edge weight parameter δ is used to remove values and obtain subnetworks. As seen in the red matrices, the values in the left matrix below a minimum edge weight of 0.01 are removed to generate the matrix on the right. The empty elements in the right matrix have the value of 0.*

2. Aim of the Project

The main goal of this project is to port HotNet2 from Python to Java. In the process of doing so, it will require understanding the two programming languages and the biological aspect of the algorithm. For Python, it will be essential to recognize the data structures and libraries used to ensure the Java implementation yields the same results. While using Java, it will be vital to use powerful enterprise-quality development and profiling tools to optimize code and address performance issues. This project will provide insight on how algorithms can be used in bioinformatics to assist in detecting subnetworks of genes related to diseases.

3. Implementation

The Java implementation was written based on Java 7 and the code is available at https://github.com/melissayan. There were two different networks used to test the HotNet2 implementation, a prototype network and the Reactome functional interaction (FI) network. The prototype network is a randomly generated small network which consists of 1000 nodes and 10,000 edges. Since the prototype network is randomly generated, the genes scores used to test the prototype were also randomly generated. To ensure that the Java implementation is capable of handling a real network, the 2015 version of the Reactome FI network was chosen. The Reactome FI is a highly reliable protein/gene network based on human curated Reactome pathways and covers close to 60% of total human genes.[11] The Reactome FI network was downloaded from the Reactome web site (http://www.reactome.org/download) and the gene scores used were from the original HotNet2 study (http://compbio-research.cs.brown.edu/pancancer/hotnet2/public/data/scores/mutation_frequency_expr_filtered.txt).[29] Running time results for the prototype network were performed by a laptop with configurations from Table 1 and results for the Reactome FI network were performed by the Oregon Health & Science University ExaCloud server.

Table 1. **Laptop Hardware Configuration**

Model	Lenovo IdeaPad Yoga 13
Operating System	Windows 10
System Type	64-bit Operating System
Processor	Intel® Core™ i7-3537U CPU @ 2.00GHz 2.50 GHz
RAM (memory)	8.00GB
Hard drive	256GB SSD

3.1 Choosing Java Libraries for Matrix Operation

The interaction network used in HotNet2 is modeled by an adjacency matrix and the majority of computation in HotNet2 is related to matrix operations (equations from Figure 7, 8, and 9). The generation of diffusion matrix was implemented in Java using the Apache Commons Math library, [30] one of Java's most popular math libraries currently available for matrix related computation. However, it still took 40 minutes to generate a 12037 by 12037 diffusion matrix for the Reactome FI network's largest component of 12037 genes.

Since the generation of diffusion matrix requires scalar multiplication, subtraction, and inversion, the performance of these three operations was gauged to find a more suitable library. Based on "Java Matrix Benchmark" results,[31] ojAlgo was the most suitable library because it is capable of handling large matrices, is the fastest for performing the inverse operation, and is also one of the fastest libraries for scalar multiplication and subtraction (Figure 10).[32] To ensure performance times would improve as expected, the time required to perform each operation in the diffusion matrix was then compared for the Apache Commons Math and ojAlgo implementations. As seen in Figure 11, the most time is spent on inversion. And the inverse operation step in creating a diffusion matrix takes the Apache Commons Math library four times as long as the ojAlgo library. By using ojAlgo, the time required to generate a diffusion matrix is reduced from 40 minutes to 10 minutes for the Reactome FI network (Figure 12). Hence, the ojAlgo library was used to generate the diffusion matrix.

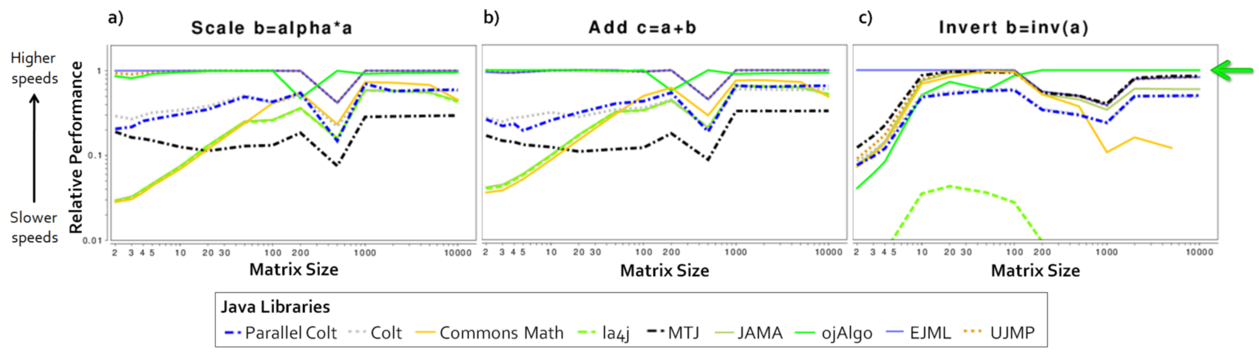


Figure 10. *Java Matrix Benchmark results for different Java libraries for scalar multiplication, subtraction, and inverse of a matrix. Since the results presented are by relative performance, Java libraries with higher speeds are near the top of the graphs. Overall, the ojAlgo library has the fastest performance time for scalar multiplication, addition, and taking the inverse of large matrices of size 10,000. This figure is modified from "Java: Basic Operation Results" from http://lessthanoptimal.github.io/Java-Matrix-Benchmark/runtime/2015_07_XeonQuad/.*

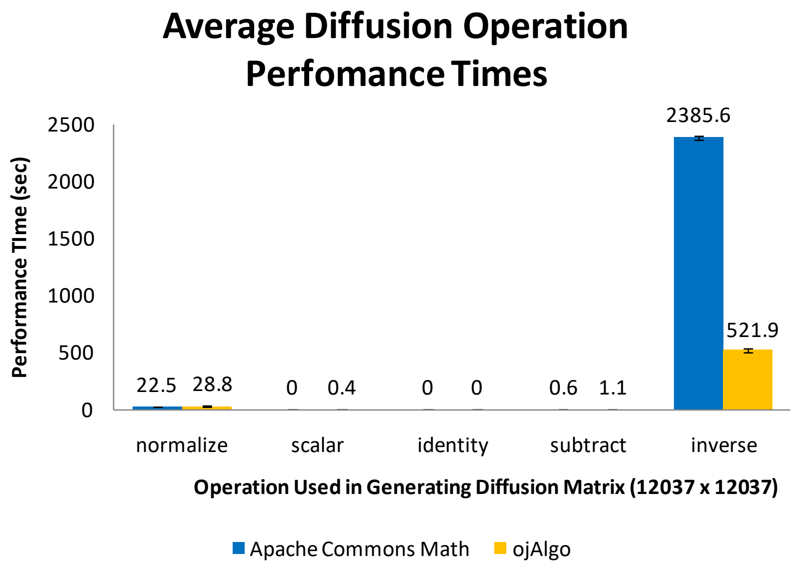


Figure 11. *Average diffusion operation performance times based on the 2015 version of the Reactome FI network. Scalar multiplication and subtraction performance times took less than 2 seconds. The most time was spent on inversion and Apache Commons Math is 1864.6 seconds slower than ojAlgo. It should be noted that although normalizing a matrix and creating an identity matrix are not operations, they were still measured to ensure generating the matrices required for the diffusion matrix did not take the most time.*

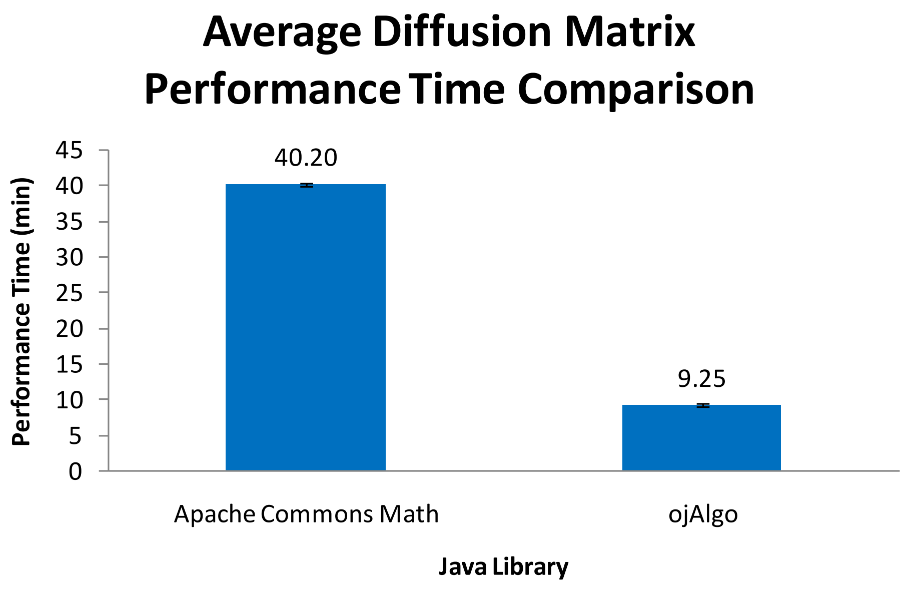


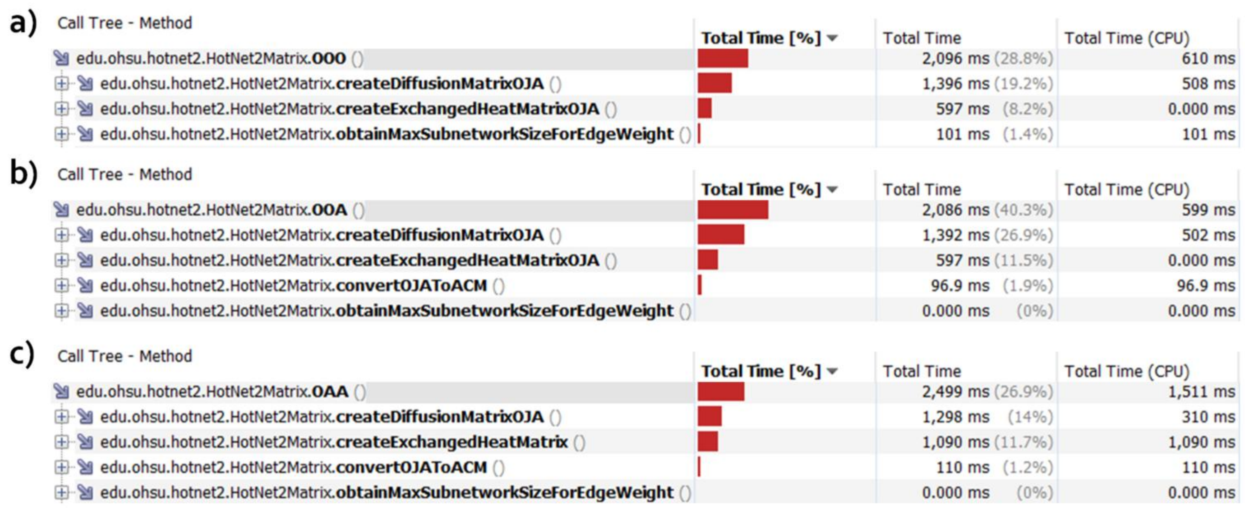
Figure 12. *Comparison of time needed to generate a diffusion matrix using Apache Commons Math versus ojAlgo over 10 different trials. On average, it takes Apache Commons Math 40 minutes to generate a diffusion matrix and it takes 10 mins for ojAlgo. It should be noted the matrices here have a dimension of 12037 x 12037 and were generated from the 2015 Reactome FI network.*

Since Apache Commons Math library was originally used to generate the diffusion matrix before switching to ojAlgo, it was still considered an option when generating the exchanged heat matrix and extracting hot subnetworks. Three different implementations were tested to determine which was most suitable for use based on performance time: OOO, OOA, and OAA (Table 2). O indicates ojAlgo was used and A indicates Apache Commons Math was used. Thus, OOO indicates the diffusion matrix, exchanged heat matrix, and matrix used for extraction all used ojAlgo. OOA specifies ojAlgo was used for the diffusion matrix and exchanged heat matrix, then converted into a matrix for use by Apache Commons Math. And OAA represents only the diffusion matrix used ojAlgo and the other steps used Apache Commons Math. Using Java VisualVM, the Java profiling tool in the Oracle JDK distribution,[33] the performance results in Figure 13 were obtained for a single trial of OOO vs OOA vs OAA. This was repeated for a total of 10 times to generate Figures 14. Based on Figure 14, OOO and

OOA have similar times and are better than OAA, so it is better to solely use the ojAlgo library instead of combining it with another library.

***Table 2.* Combination of Different Matrix Libraries for HotNet2**

	Diffusion Matrix		Exchanged Heat Matrix		Subnetwork Identification	
Abbreviation	ojAlgo	Apache Commons Math	ojAlgo	Apache Commons Math	ojAlgo	Apache Commons Math
OOO	x		x		x	
OOA	x		x			x
OAA	x			x		x


a)

Call Tree - Method	Total Time [%]	Total Time	Total Time (CPU)
edu.ohsu.hotnet2.HotNet2Matrix.**OOO** ()		2,096 ms (28.8%)	610 ms
edu.ohsu.hotnet2.HotNet2Matrix.**createDiffusionMatrixOJA** ()		1,396 ms (19.2%)	508 ms
edu.ohsu.hotnet2.HotNet2Matrix.**createExchangedHeatMatrixOJA** ()		597 ms (8.2%)	0.000 ms
edu.ohsu.hotnet2.HotNet2Matrix.**obtainMaxSubnetworkSizeForEdgeWeight** ()		101 ms (1.4%)	101 ms

b)

Call Tree - Method	Total Time [%]	Total Time	Total Time (CPU)
edu.ohsu.hotnet2.HotNet2Matrix.**OOA** ()		2,086 ms (40.3%)	599 ms
edu.ohsu.hotnet2.HotNet2Matrix.**createDiffusionMatrixOJA** ()		1,392 ms (26.9%)	502 ms
edu.ohsu.hotnet2.HotNet2Matrix.**createExchangedHeatMatrixOJA** ()		597 ms (11.5%)	0.000 ms
edu.ohsu.hotnet2.HotNet2Matrix.**convertOJAToACM** ()		96.9 ms (1.9%)	96.9 ms
edu.ohsu.hotnet2.HotNet2Matrix.**obtainMaxSubnetworkSizeForEdgeWeight** ()		0.000 ms (0%)	0.000 ms

c)

Call Tree - Method	Total Time [%]	Total Time	Total Time (CPU)
edu.ohsu.hotnet2.HotNet2Matrix.**OAA** ()		2,499 ms (26.9%)	1,511 ms
edu.ohsu.hotnet2.HotNet2Matrix.**createDiffusionMatrixOJA** ()		1,298 ms (14%)	310 ms
edu.ohsu.hotnet2.HotNet2Matrix.**createExchangedHeatMatrix** ()		1,090 ms (11.7%)	1,090 ms
edu.ohsu.hotnet2.HotNet2Matrix.**convertOJAToACM** ()		110 ms (1.2%)	110 ms
edu.ohsu.hotnet2.HotNet2Matrix.**obtainMaxSubnetworkSizeForEdgeWeight** ()		0.000 ms (0%)	0.000 ms

Figure 13. *VisualVM performance results from running 3 separate possible implementations of the matrices used in HotNet2 algorithm on a prototype network of 1000 nodes and 5000 edges. VisualVM is a profiling tool with the ability to track application performance by providing a method's execution time. The three different tested combinations are: OOO, OOA, and OAA. (a) OOO indicates that the diffusion matrix and exchanged heat matrix were both created using ojAlgo and delta selection was made using an ojAlgo exchanged heat matrix; (b) OOA indicates ojAlgo was used to create the diffusion matrix and exchanged heat matrix, then converted into an Apache Commons Math matrix for delta selection; (c) OAA indicates ojAlgo generated a diffusion matrix which was converted into an Apache Commons Math matrix for creating a exchanged heat matrix and delta selection. It should be noted that the figure is a combination of the three parts which were tested separately.*

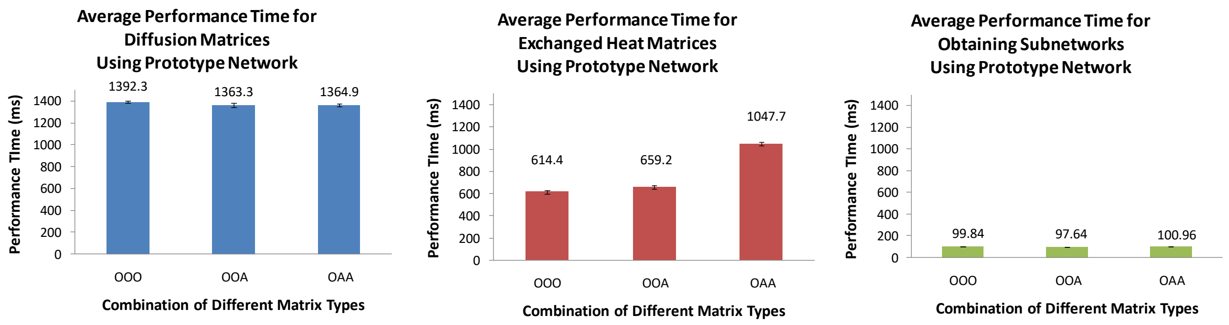


Figure 14. *Average performance time for a combination of different matrix libraries used in the HotNet2 algorithm on a prototype network of 1000 nodes and 5000 edges based on 10 trials. For the three different tested combinations, O indicates ojAlgo was used and A indicates Apache Commons Math was used (a detailed description can be found in Table 2). The execution time required to generate a diffusion matrix and obtain subnetworks was similar for the two libraries. However, when generating an exchanged matrix, it took the Apache Commons Math library much longer than 660 ms.*

3.2 Implementation of the HotNet2 Algorithm in Java

An overview of steps required to run the HotNet2 algorithm is shown in Figure 15.

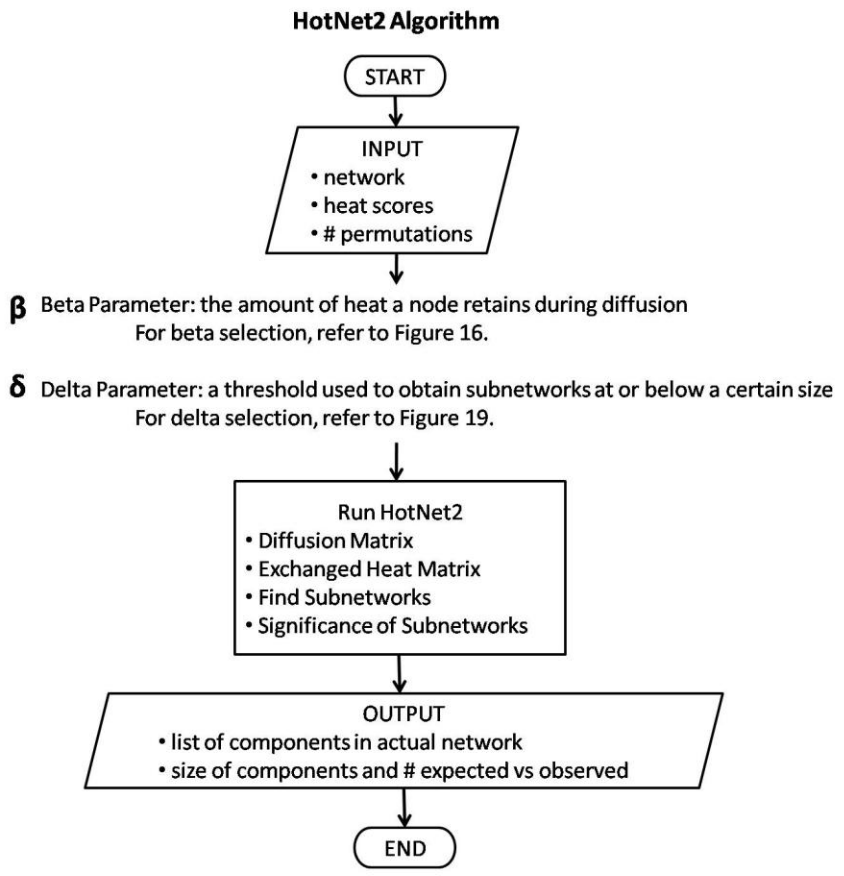

HotNet2 Algorithm
START
INPUT
• network
• heat scores
• # permutations
β Beta Parameter: the amount of heat a node retains during diffusion
For beta selection, refer to Figure 16.
δ Delta Parameter: a threshold used to obtain subnetworks at or below a certain size
For delta selection, refer to Figure 19.
Run HotNet2
• Diffusion Matrix
• Exchanged Heat Matrix
• Find Subnetworks
• Significance of Subnetworks
OUTPUT
• list of components in actual network
• size of components and # expected vs observed
END

Figure 15. *Flowchart of the HotNet2 algorithm. Details on beta and delta selection can be found in Figure 16 and 19, respectively.*

After the diffusion matrix, an exchanged heat matrix is created for extraction of hot subnetworks (Figure 8 and 9). The pseudocode for generating an exchanged heat matrix is provided below:

```
Input: Diffusion matrix $F$, set of genes $G$, and heat scores $S$
Output: Exchanged heat matrix
1. $M \leftarrow$ Map(key: null, value: null)
2. For each gene in $G$ do
3.     If gene has $S$ then $M \leftarrow$(key: gene, value: $S$) else $M \leftarrow$(key: gene, value: 0.0)
4. $D_{\vec{h}} \leftarrow$ DiagonalMatrixWithHeatScores
5. For each gene in F do
6.     $s \leftarrow$ get gene's value from $M$
7.     $D_{\vec{h}\,(i,i)} \leftarrow (s)$
8. Multiply $F$ by $D_{\vec{h}}$
```

To generate a directed weighted graph, elements in the exchanged heat matrix at or above the minimum edge weight δ threshold are used to generate a new graph. From the newly generated graph, only strongly connected components are extracted and identified as subnetworks of potential interest.

3.3 Parameter Selection

There are two parameters required for the HotNet2 algorithm, beta and delta.[29] Beta determines the amount of heat a node will retain during diffusion and delta is a threshold used to obtain subnetworks at or below a certain size.

The process required to select beta is illustrated in Figure 16. In order to select beta, the betweenness centrality of all proteins was calculated to determine five "source proteins" to assess the influence these proteins have on all other proteins in the network.[18,29] These five "source proteins" were selected based on the following five betweenness centrality scores: minimum, 25% quantile, median, 75% quantile, and maximum. In addition to these five proteins, TP53 was also used as a source protein for comparing with reported results from the original Python implementation. During the process of selecting beta, 20 different diffusion matrices were generated using beta values from 0.05 to 1.00 (ex. $\beta = 0.05, 0.10, 0.15 \dots 1$). For each diffusion matrix, how the heat within the "source proteins" spread to direct neighboring proteins, secondary neighboring proteins, and all other proteins was observed. For instance, for each protein, influence values (ex. 0.001, 0.002, 0.003 … 0.1) were used as a threshold to determine how many of those proteins in the diffusion matrix had a value greater than the given influence value. From the observed values, a beta value graph similar to Figure 17a was plotted for each protein and the given beta value. Then, for each of the six proteins an inflection point diagram was compiled based on the inflection point for direct neighboring proteins from the different beta graphs and the initial maximum inflection point was selected as the beta parameter (Figure 17b).

The Java implementation for beta selection was then validated with the Supplementary Figure 24 results from Leiserson et al.,[29] using the TP53 gene from the iRefIndex network because no source code for obtaining the beta parameter is available. In Figure 18, it shows the Java implementation yielded the same results as the results seen in Supplementary Figure 24. However, it should be noted that after the HotNet2 paper was published new source code was released on May 31, 2016 with a bug fix in generating the diffusion matrix. Accordingly, the Java code for generating diffusion matrices was updated. However, results and Python code for beta selection are not available in the updated Python HotNet2 code to enable verification of the current Java implementation. But based on this comparison result (Figure 18), we believe our Java implementation should be correct.

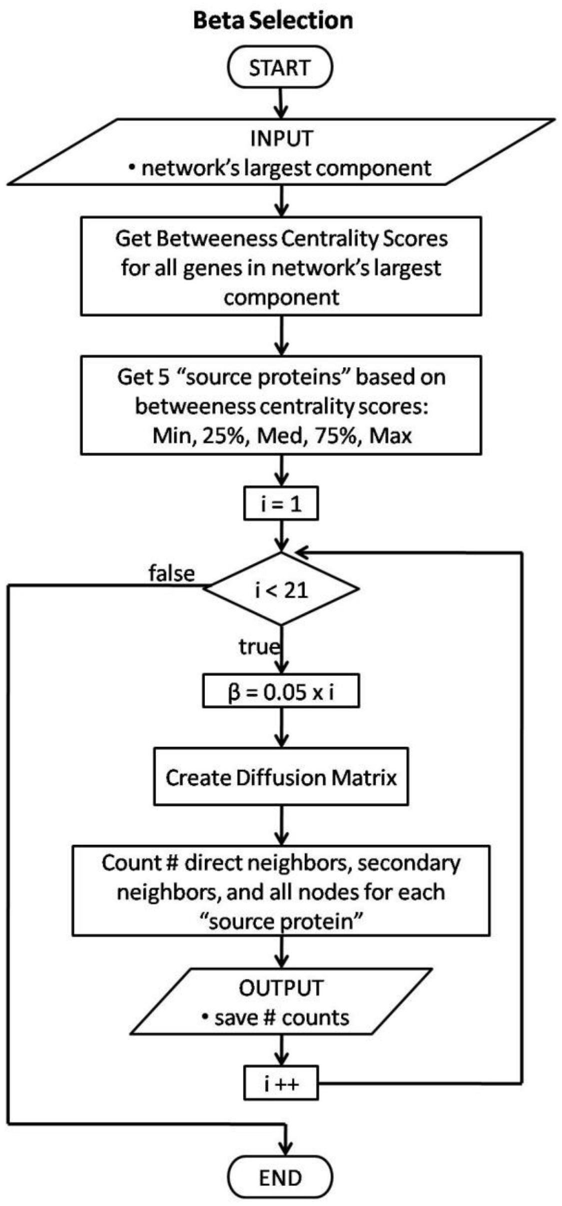

Beta Selection
START
INPUT
• network's largest component
Get Betweeness Centrality Scores for all genes in network's largest component
Get 5 "source proteins" based on betweeness centrality scores: Min, 25%, Med, 75%, Max
i = 1
false
i < 21
true
β = 0.05 x i
Create Diffusion Matrix
Count # direct neighbors, secondary neighbors, and all nodes for each "source protein"
OUTPUT
• save # counts
i ++
END

Figure 16. *Flowchart for beta selection. Using the largest component in a network, betweenness centrality scores are calculated to get 5 "source proteins". Then a range of β values from 0.05 to 1.00 are used to obtain results for each source protein based on the number of direct neighbors, secondary neighbors, and number of all other proteins in the network. The results are then saved in a file to be converted into a graph as seen in Figure 18 (see below).*

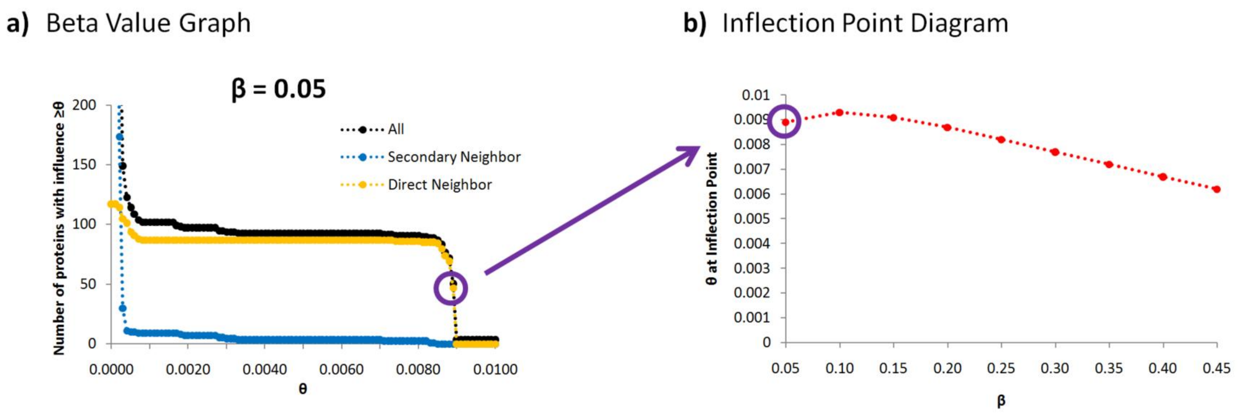


Figure 17. *An example illustrating how to create an inflection point diagram using beta value graphs. (a) A beta value graph for a source protein was generated using β=0.05. And additional similar beta value graphs will be generated based on different beta values. The purple circle indicates where the inflection point for the direct neighboring protein is. (b) By using the inflection point detected in each of the beta value graphs, an inflection point diagram can be generated. For instance, in the beta value graph for β=0.05 the purple circled inflection point was 0.009, so in the inflection point diagram that point is plotted over.*

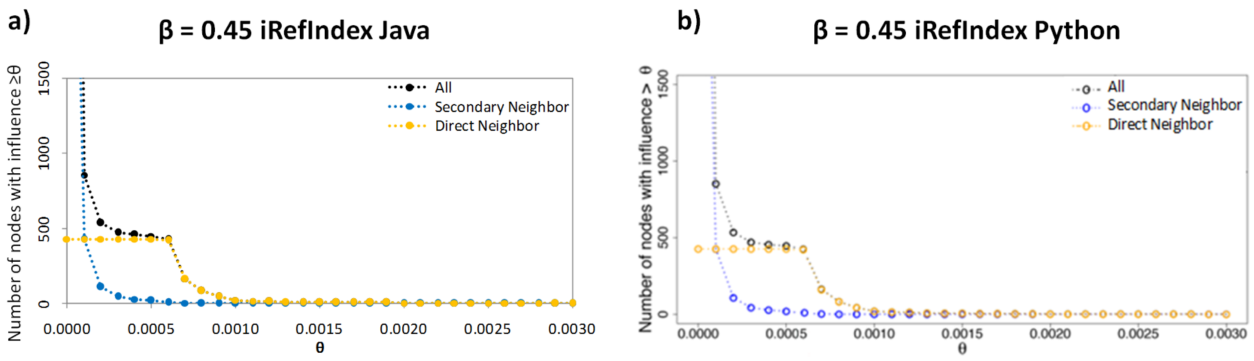


Figure 18. *Comparison of Java and Python results on beta selection using the TP53 gene in the iRefIndex network. The Java implementation on the left matches the Python results on the right. The black dotted line represents all genes within the network, the blue dotted line represents secondary neighbor nodes that are 2 interactions away from the TP53 gene, and the yellow dotted line represents direct neighbor nodes. Figure b is adapted from Supplementary Figure 24* from Leiserson et al..[29]

The delta parameter is chosen to ensure that large subnetworks found in the actual network are not likely to be found in random networks. To obtain delta, 100 random networks are generated using the switch algorithm from Python's NetworkX package.[34] Using the switch algorithm ensures that the graph is still a single connected component, containing the same number of nodes that have the same degrees as in the original network. From each random network, four minimum deltas are chosen when all the strongly connected component sizes obtained are less than or equal to a given subnetwork size of 5, 10, 15, and 20, as seen in Figure 19. For each of the given subnetwork sizes, there will be a total of 100 minimum deltas and the median delta value from the 100 minimum deltas will be chosen as the delta for that subnetwork size. So subnetworks of size 5, 10, 15, and 20 will each have their own individual selected delta value. Each of the four selected delta values is used to find p-values of significant subnetworks sizes from two to ten, then the delta value with the most significant p-values less than 0.05 is chosen as the delta (an example is shown in Table 3) . The Java implementation was then validated with the Python implementation to ensure correctness. As seen in Figure 20, the Java results for selecting delta values correspond very closely with the Python results.

***Table 3.* iRefIndex Delta Selection using MutSigCV –log$_{10}$ q-value Heat Scores.**

		Delta			
		0.016167	**0.016871**	**0.020113**	**0.02973**
Subnetwork Size	**2**	0.363	0.339	0.08	0.111
	3	0.15	0.122	0.155	0.004
	4	0.294	0.278	0.024	0.001
	5	0.407	0.362	0.043	0
	6	0.242	0.194	0.007	0
	7	0.139	0.113	0.004	0
	8	0.077	0.067	0.002	0.001
	9	0.049	0.045	0	0
	10	0.026	0.024	0	0

*Note: The delta value in green is the selected value used for analysis because it has the most significant p-values (p-value < 0.05; highlighted in yellow) for the subnetwork sizes. The p-values are obtained from the HotNet2 pan-cancer analysis website from Leiserson et al.,[29] at http://compbio-research.cs.brown.edu/pancancer/hotnet2/public/data/runs/mutsigcv/iref/.

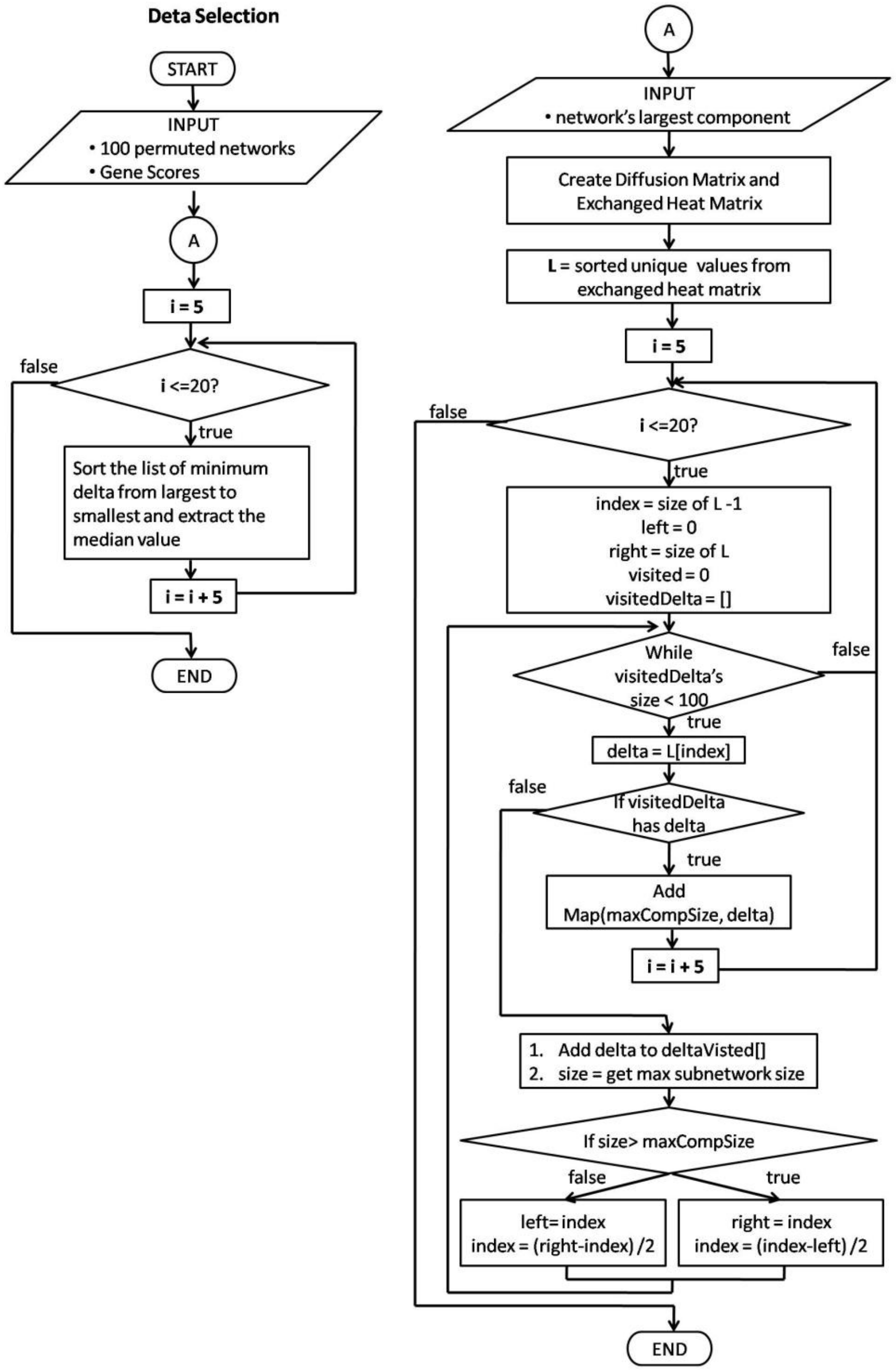

Deta Selection
START
INPUT
• 100 permuted networks
• Gene Scores
A
i = 5
i <=20?
false
true
Sort the list of minimum delta from largest to smallest and extract the median value
i = i + 5
END
A
INPUT
• network's largest component
Create Diffusion Matrix and Exchanged Heat Matrix
L = sorted unique values from exchanged heat matrix
i = 5
i <=20?
false
true
index = size of L -1
left = 0
right = size of L
visited = 0
visitedDelta = []
While visitedDelta's size < 100
false
true
delta = L[index]
If visitedDelta has delta
false
true
Add Map(maxCompSize, delta)
i = i + 5
1. Add delta to deltaVisted[]
2. size = get max subnetwork size
If size> maxCompSize
false
true
left= index
index = (right-index) /2
right = index
index = (index-left) /2
END

Figure 19. *Flowchart for delta selection. For each of the 100 random networks generated, an exchanged heat matrix is generated and a sorted list of all unique edge weight values is extracted from the matrix. Using binary search, the minimum delta is obtained from the list for each of the maximum subnetwork sizes of 5, 10, 15, and 20. Then the minimum deltas are collected from 100 random networks, sorted from largest to smallest and the median value for each size is returned.*

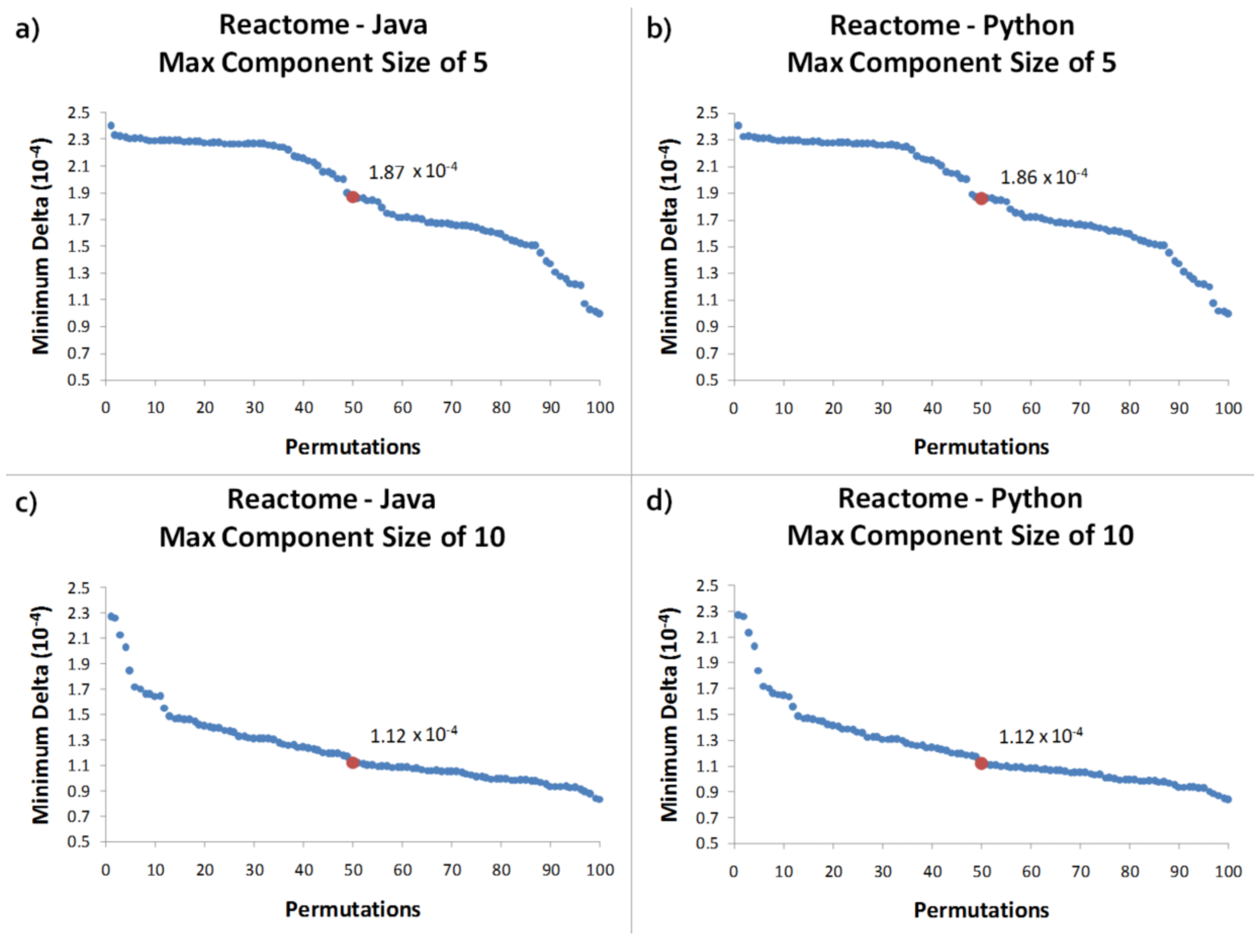


Figure 20. *Comparison of the select delta Java and Python results in the Reactome FI network using β=0.25. Results from the Java implementation match with ones from the Python implementation. (a-b) For the maximum component size of 5, the scatterplot generated by the minimum edge weight values follows a similar trend and the median minimum edge weight is close for both. (c-d) A similar trend line also exists when the max component size is 10 and the median minimum edge weight for both is 1.12 x 10*$^{-4}$.

4. Application of Java HotNet2

The Reactome FI network was chosen to provide users a real example for the HotNet2 algorithm process shown in Figure 15. When using a network for the first time, the preferred method is to run beta selection instead of randomly choosing a beta parameter. From the beta selection text file output, the graphs from Figure 22a – 22e can be generated for a range of possible beta parameters. Through combining the inflection point results for each beta parameter from 0.05 to 1.00, a comprehensive diagram (Figure 22f) is obtained and the suggested beta parameter is obtained by selecting the beta where the highest inflection point is first reached. In Figure 22, the results were only for TP53, but the actual results from beta selection will also include the results to obtain the graphs for 5 “source proteins” as seen in Figure 23. Since 0.25 was the inflection point for the 75% quantile and maximum betweenness centrality of proteins and it is also close to TP53’s beta parameter of 0.20, the beta parameter of 0.25 was determined to be most suitable for the Reactome FI network. Using the beta parameter of 0.25, four different delta parameters were selected for subnetwork sizes less than or equal to 5, 10, 15, and 20. Figure 24 depicts graphs the delta selection output and the red dot is the median minimum selected delta parameter. A combination of the recommended beta parameter and the selected delta parameter will then be used to identify subnetworks when executing the HotNet2 algorithm.

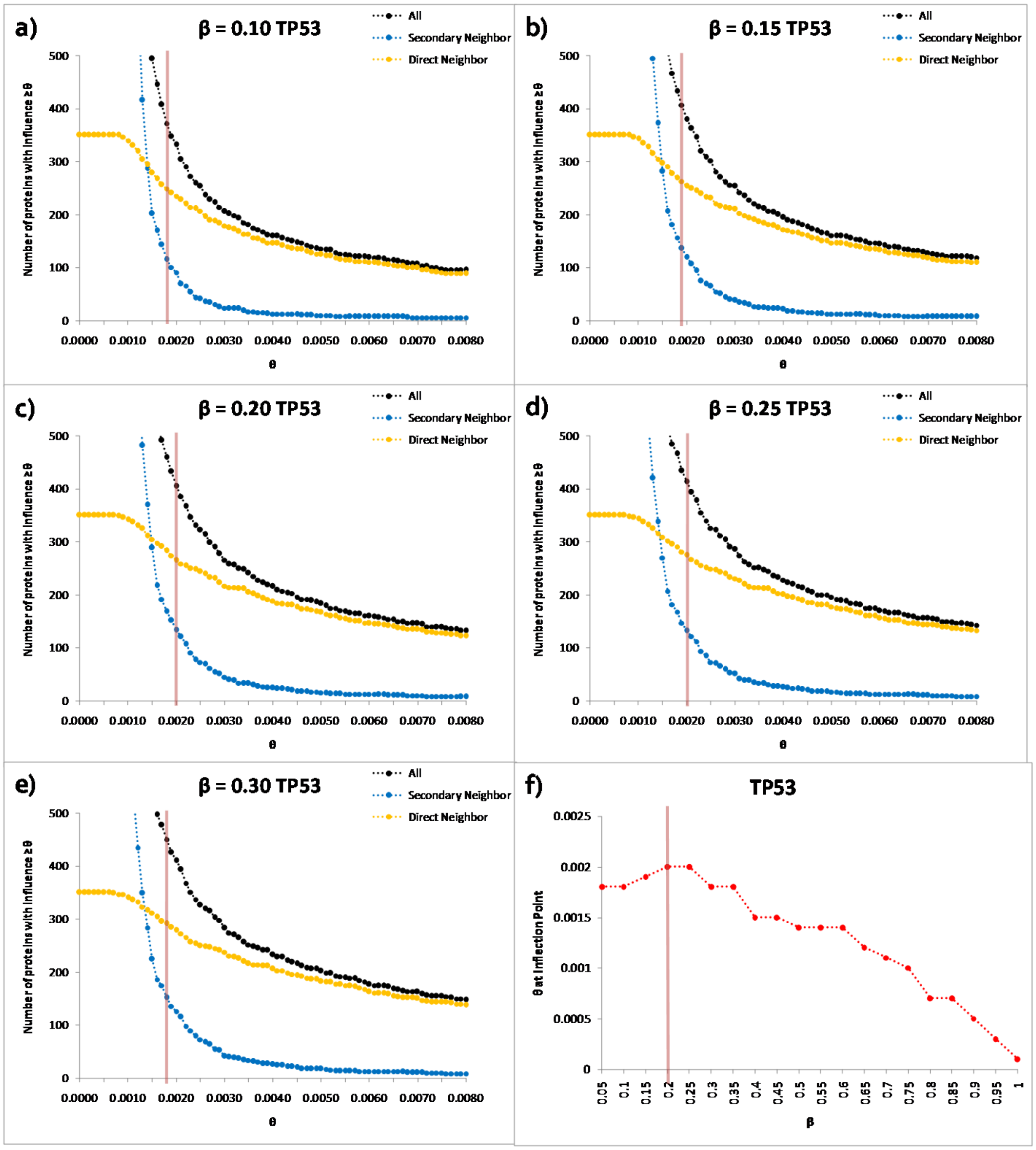


Figure 22. *Beta selection results using the Reactome FI network and the TP53 gene. The results obtained from a range of beta values (a-e) were used to find inflection points. Then an inflection point diagram (f) was compiled by combining the inflection points for each beta into one graph. It shows 0.20 as the suggested beta, because that is the point before heat from a protein starts diffusing more to a direct neighbor's neighbor instead of the protein's direct neighbor.*

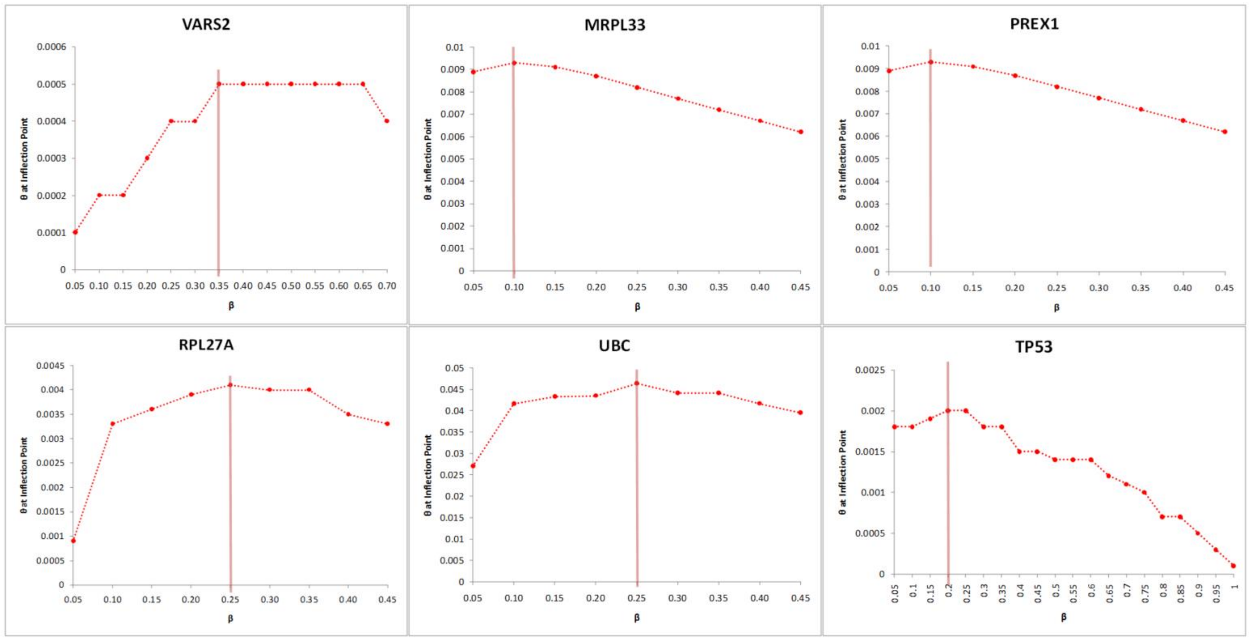


Figure 23. *Beta selection inflection point diagram results using the Reactome FI network. The 5 "source proteins" chosen based on the minimum, 25% quantile, median, 75% quantile, and maximum betweenness centrality scores are as follows: VARS2, MRPL33, PREX1, RPL27A, and UBC.*

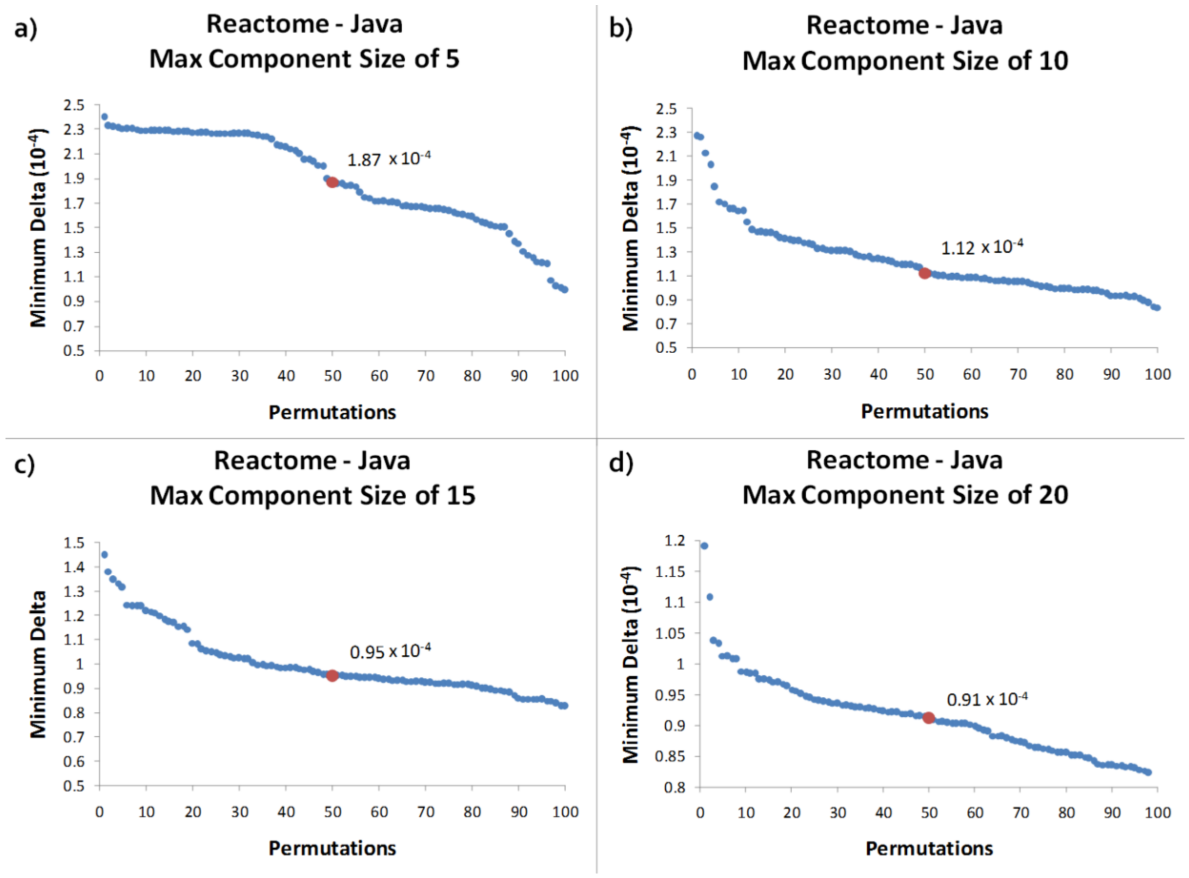


Figure 24. *Delta selection results using the Reactome FI network and $\beta=0.25$. The median minimum delta for maximum component sizes of 5, 10, 15, and 20 are as follows: 1.87×10^{-4}, 1.12×10^{-4}, 0.95×10^{-4}, and 0.91×10^{-4}.*

5. Future Work

The current Java implementation of HotNet2 uses pure Java libraries. It is possible to improve the performance time of matrix operations even further by using native libraries via JNI.[35] An alternative to using the pure Java ojAlgo library is using the Matrix-Toolkits-Java library (MTJ-N).[36] Based on the results in “Java Matrix Benchmark”, using MTJ-N would improve the performance time for the inversion and multiplication operations.[31] Although MTJ-N is slower than ojAlgo for subtraction and scalar multiplication, the actual latency time will vary little and most time will be spent on inversion (as seen in Figure 12).

Since the HotNet2 algorithm has been ported from Python to HotNet2, it can be integrated into Java-based standalone applications. In particular, it would be possible to implement HotNet2 into ReactomeFIViz,[37] a Cytoscape app,[38] to allow researchers and clinicians to use this powerful diffusion-based algorithm alongside pre-existing network and pathway analysis features in a graphical user interface (GUI) setting for the entire workflow for the first time. Furthermore, integration into the open source software tool Cytoscape ensures that this tool will be accessible to a large user base because it is widely used in the research community and has 10,000 downloads per year.[38]

6. References

1. Mitra, K., Carvunis, A.-R., Ramesh, S. K. & Ideker, T. Integrative approaches for finding modular structure in biological networks. *Nat. Rev. Genet.* **14,** 719–732 (2013).
2. Joyce, A. R. & Palsson, B. O. The model organism as a system: integrating 'omics' data sets. *Nat. Rev. Mol. Cell Biol.* **7,** 198–210 (2006).
3. Wang, Y.-C. & Chen, B.-S. Integrated cellular network of transcription regulations and protein-protein interactions. *BMC Syst. Biol.* **4,** 20 (2010).
4. Sharan, R., Ulitsky, I. & Shamir, R. Network-based prediction of protein function. *Mol. Syst. Biol.* **3,** 13 (2007).
5. Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N. & Barabási, A.-L. The large-scale organization of metabolic networks. *Nature* **407,** 651–654 (2000).
6. Ideker, T., Ozier, O., Schwikowski, B. & Siegel, A. F. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics* **18 Suppl 1,** S233–40 (2002).
7. Fortney, K., Kotlyar, M. & Jurisica, I. Inferring the functions of longevity genes with modular subnetwork biomarkers of Caenorhabditis elegans aging. *Genome Biol.* **11,** R13 (2010).
8. Alon, U. Network motifs: theory and experimental approaches. *Nat. Rev. Genet.* **8,** 450–61 (2007).
9. Goh, K.-I., Cusick, M. E., Valle, D., Childs, B., Vidal, M. & Barabási, A.-L. The human disease network. *Proc. Natl. Acad. Sci. U. S. A.* **104,** 8685–8690 (2007).
10. Barabási, A.-L., Gulbahce, N. & Loscalzo, J. Network medicine: a network-based approach to human disease. *Nat. Rev. Genet.* **12,** 56–68 (2011).

11. Wu, G., Feng, X. & Stein, L. A human functional protein interaction network and its application to cancer data analysis. *Genome Biol* **11,** R53 (2010).

12. Chen, H., Zhu, Z., Zhu, Y., Wang, J., Mei, Y. & Cheng, Y. Pathway mapping and development of disease-specific biomarkers: Protein-based network biomarkers. *J. Cell. Mol. Med.* **19,** 297–314 (2015).

13. Ahn, A. C., Tewari, M., Poon, C.-S. & Phillips, R. S. The Limits of Reductionism in Medicine: Could Systems Biology Offer an Alternative? *PLoS Med.* **3,** e208 (2006).

14. Chan, S. Y. & Loscalzo, J. The Emerging Paradigm of Network Medicine in the Study of Human Disease. *Circ. Res.* **111,** 359–374 (2012).

15. Tillmann, T., Gibson, A. R., Scott, G., Harrison, O., Dominiczak, A. & Hanlon, P. Systems Medicine 2.0: Potential Benefits of Combining Electronic Health Care Records With Systems Science Models. *J. Med. Internet Res.* **17,** e64 (2015).

16. Arrell, D. K. & Terzic, a. Network systems biology for drug discovery. *Clin. Pharmacol. Ther.* **88,** 120–125 (2010).

17. Wu, G. & Stein, L. A network module-based method for identifying cancer prognostic signatures. *Genome Biol.* **13,** R112 (2012).

18. Newman, M. E. . *Networks An Introduction*. (Oxford University Press, 2010).

19. Navlakha, S. & Kingsford, C. The power of protein interaction networks for associating genes with diseases. *Bioinformatics* **26,** 1057–1063 (2010).

20. Zhu, J., Qin, Y., Liu, T., Wang, J. & Zheng, X. Prioritization of candidate disease genes by topological similarity between disease and protein diffusion profiles. *BMC Bioinformatics* **14 Suppl 5,** S5 (2013).

21. Vanunu, O., Magger, O., Ruppin, E., Shlomi, T. & Sharan, R. Associating Genes and

Protein Complexes with Disease via Network Propagation. *PLoS Comput. Biol.* **6,** e1000641 (2010).

22. Köhler, S., Bauer, S., Horn, D. & Robinson, P. N. Walking the Interactome for Prioritization of Candidate Disease Genes. *Am. J. Hum. Genet.* **82,** 949–958 (2008).

23. Van Allen, E. M., Wagle, N., Stojanov, P., Perrin, D. L., Cibulskis, K., Marlow, S., Jane-Valbuena, J., Friedrich, D. C., Kryukov, G., Carter, S. L., McKenna, A., Sivachenko, A., Rosenberg, M., Kiezun, A., Voet, D., Lawrence, M., Lichtenstein, L. T., Gentry, J. G., Huang, F. W., Fostel, J., Farlow, D., Barbie, D., Gandhi, L., Lander, E. S., Gray, S. W., Joffe, S., Janne, P., Garber, J., MacConaill, L., Lindeman, N., Rollins, B., Kantoff, P., Fisher, S. A., Gabriel, S., Getz, G. & Garraway, L. A. Whole-exome sequencing and clinical interpretation of formalin-fixed, paraffin-embedded tumor samples to guide precision cancer medicine. *Nat. Med.* **20,** 682–688 (2014).

24. Ding, L., Wendl, M. C., Koboldt, D. C. & Mardis, E. R. Analysis of next-generation genomic data in cancer: accomplishments and challenges. *Hum. Mol. Genet.* **19,** R188–96 (2010).

25. Wheeler, D. a & Wang, L. From human genome to cancer genome: the first decade. *Genome Res.* **23,** 1054–62 (2013).

26. Garraway, L. A. & Lander, E. S. Lessons from the cancer genome. *Cell* **153,** 17–37 (2013).

27. Vandin, F., Upfal, E. & Raphael, B. J. Algorithms for detecting significantly mutated pathways in cancer. *J. Comput. Biol.* **18,** 507–22 (2011).

28. Vandin, F., Clay, P., Upfal, E. & Raphael, B. J. Discovery of mutated subnetworks associated with clinical data in cancer. *Pac. Symp. Biocomput.* 55–66 (2012). doi:9789814366496_0006

29. Leiserson, M. D. M., Vandin, F., Wu, H.-T., Dobson, J. R., Eldridge, J. V, Thomas, J. L., Papoutsaki, A., Kim, Y., Niu, B., McLellan, M., Lawrence, M. S., Gonzalez-Perez, A., Tamborero, D., Cheng, Y., Ryslik, G. A., Lopez-Bigas, N., Getz, G., Ding, L. & Raphael, B. J. Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes. *Nat. Genet.* **47,** 106–114 (2015).

30. Apache Software Foundation. Commons Math: The Apache Commons Mathematics Library. at <http://commons.apache.org/>

31. Abeles, P. Java Matrix Benchmark. (2015). at <http://lessthanoptimal.github.io/Java-Matrix-Benchmark/>

32. Optimatika. oj! Algorithms. at <http://ojalgo.org/>

33. Sedlacek, J. & Hurka, T. VisualVM. at <visualvm.java.net>

34. Hagberg, A. A., Schult, D. A. & Swart, P. J. Exploring network structure, dynamics, and function using NetworkX. *Proc. 7th Python Sci. Conf.* 11–15 (2008). at <http://networkx.readthedocs.io/en/stable/reference/citing.html>

35. Oracle. Java Native Interface. *ORACLE*. at <http://docs.oracle.com/javase/7/docs/technotes/guides/jni/>

36. Halliday, S. & Heimsund, B.-O. Matrix-Toolkits-Java. at <https://github.com/fommil/matrix-toolkits-java>

37. Wu, G., Dawson, E., Duong, A., Haw, R. & Stein, L. ReactomeFIViz: a Cytoscape app for pathway and network-based data analysis. *F1000Research* **3,** 146 (2014).

38. Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B. & Ideker, T. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res.* **13,** 2498–504 (2003).